Enterprise Architecture Approach to Build API Economy

Mujahid Sultan
Dept. of Computer Science
Ryerson University
Toronto, Canada
mujahid.sultan@ryerson.ca

Daya Rajaratnam
Aytra Inc.
Toronto, Canada
daya@aytra.com

Kam Patel
Aytra Inc.
Toronto, Canada
kam.patel@aytra.com

Abstract— **The IT industry has seen many transformations in the Software Development Life Cycle (SDLC) methodologies and development approaches. SDLCs range from waterfall to agile, and the development approaches from monolith to microservices. The recent advances in cloud-native microservices-based development approaches have created an API economy in the modern enterprise, the APIs act as contracts and carry business value. This publication describes an approach to designing microservices and APIs based on stakeholder needs. It describes an enterprise architecture framework for microservices-based cloud-native developments.**

Keywords— Enterprise Architecture, APIs, Microservices, API Economy, DevOps, Requirements Engineering, Stakeholder Viewpoints

I. Introduction

Widespread adoption of agile project management, independent delivery with microservices, and automated deployment with DevOps has tremendously sped up the systems development. The real game-changer is continuous integration (CI) and continuous deployment (CD). Organizations can do multiple releases a day, known as continuous delivery, shortening the test, release, and deployment cycles from weeks to minutes. The Maturity of container technologies like Docker and container orchestration platforms like Kubernetes has promoted microservices architecture, especially in cloud-native developments. Various tools are available for setting up CI/CD pipelines. While organizations are moving away from monolith applications and moving towards microservices-based architectures, they can quickly accumulate hundreds of such microservices. Microservice are provisioned and accessed via application programming interfaces (APIs). While CI/CD tools and DevOps processes offer speed and time to market, microservices reusability, life cycle management and monetization may not be materialized unless microservices and APIs are linked to enterprise-wide stakeholders' needs. The link between stakeholders' needs and microservices/APIs is not well captured nor adequately defined. This paper describes a structured method to create a logical link between microservices-based agile developments and enterprise stakeholders' viewpoint-concerns. This method enables capturing and documenting enterprise-wide stakeholders' needs, whether these stakeholders are business owners, planners (product owners, architects), designers (developers, DevOps engineers), or the partners and subscribers of an enterprise. The taxonomies created with this method enable enterprises to effectively associate business value to API's, manage life-cycle, use management tools, prescribe rate limits and govern APIs. Thus, it enables the API economy in an enterprise.

II. Motivation

Business process management (BPM) and Enterprise Architecture (EA) disciplines emerged to streamline the business and systems gap. With the adoption of agile methodologies and DevOps, this gap is widening.

EA management is a practice to document relationships among businesses and systems. Many EA frameworks have evolved [1] and have received some maturity [2] over the past decade, TOGAF, FEAF, and Zachman, to name a few. A comprehensive review of EA and EA frameworks is given in [3, 4, 5], and an evaluation of EA frameworks is given in [6, 7, 8]. EA frameworks consist of artifacts, which are descriptions of the enterprise from a specific viewpoint of a group of stakeholders [9, 10]. The stakeholders' groups are owners, designers (architects), systems engineers, developers and DevOps.

Zachman introduced the concept of Information System Architecture (ISA) in 1987 [11]. The Zachman framework describes stakeholders' views which focuses on five "wh" interrogatives (what, who, where, why, and when) and one "h" interrogative (how). This focus comes from journalism's w5h theory [12].

Sultan and Miranskyy applied linguistic findings to establish that the w5h set of interrogatives is not complete and added the interrogative "which" [13, 14]. They denoted this new set of seven interrogatives as w6h. They demonstrated that asking questions in a prescribed order improves information flow for describing stakeholders' viewpoint-concerns.

The fundamental difference between the monolith and microservices-based architectures is the codebase. Monolith applications work with a central and relatively large codebase, whereas microservices have an independent and relatively small codebase. Due to the agile SDLCs, new microservices can be designed quickly and immediately deployed in production. Agile management practices and microservice designs culminated in a new way of developing software - the DevOps, where CI and CD play a significant role. While EA frameworks for monolith applications have matured, there is no structured approach for defining EA for microservices-based developments. This paper extends the w6h EA framework and w6h requirements engineering

pattern developed by Sultan and Merensky [13, 14] to microservices-based developments.

Roadmap: The paper is organized as follows. Section III describes methods to develop an architecture framework for microservices-based developments. The proposed framework is given in Section IV. In Section V, we discuss the framework

III. Methods

ANSI/IEEE standard 1471, Recommended Practice for Architecture Descriptions of Software-Intensive Systems, defines Architecture as the fundamental organization of a system embodied in its components, their relationship to each other and the environment, and the principles guiding its design and evolution. Architectural Description (AD) is a collection of products to document architecture. A view is defined as representing a whole system from the perspective of a related set of stakeholders' concerns. A viewpoint defines the perspective from which a view is taken. A viewpoint is where a person is looking from - the vantage point or concern that determines what stakeholders need is, and a view is what stakeholders see.

To describe and capture the stakeholders' viewpoint-concerns for microservices, we use the grouping of stakeholders' views as defined by Zachman [11] and given below:

A) Scope (Ballpark View),
B) Business Model (Owner's View),
C) System Model (Designer's View),
D) Technology Model (Builder's View),
E) Detailed Representations (Subcontractor's View).

In the following, we define each view and the group of stakeholders who hold this view. Then we briefly describe stakeholders' viewpoint-concerns from each of w6h viewpoints of an enterprise for microservice-based developments. These descriptions constitute EA from a microservices perspective in the order given by Sultan and Merensky [14].

A. Scope (Ballpark View)

"Ballpark View" sets the scope and puts architecture effort in perspective; and is also called the 'contextual' view of the organization and describes the environment and surroundings the enterprise will function in and interact with. The following groups of stakeholders see the organization from this view: business development directors, delivery managers, CIOs, CFOs, CSOs, etc. Below is a detailed description of their viewpoint-concerns from each viewpoint.

These stakeholders need to capture the contextual Information of APIs and Microservices (to be provisioned by the organization).

who (people):
This stakeholders' *viewpoint-concern* captures the list of businesses an enterprise interacts with and may expose microservices too. The list of the organizations an enterprise might interchange information via APIs. This viewpoint-concern is used in planning for the business to business (B2B) and business to consumer (B2C) relationships.

what (data):
This stakeholders' *viewpoint-concern* captures a list of entities an enterprise needs to conduct business with other businesses and organizations - used in the design of microservices and may be required to offer APIs.

which (selection):
In this *viewpoint-concern*, we need to capture and select the list of organizations critical to the enterprise - the target audiences of microservices and APIs important to business or vice versa - select microservices and APIs to conduct business.

where (network):
This *viewpoint-concern* captures locations where the enterprise conducts its business - the candidate locations for microservices. The list of partner locations is also of interest from this viewpoint for planning purposes.

how (function):
Defining the business processes required for microservices and APIs at the contextual level is essential. This allows for strategic planning and setting future goals at the highest level of the enterprise hierarchy.

why (motivation):
It becomes crucial to link the vision and business reasons for providing microservices and APIs at the contextual level. IT may provide services and offer APIs without business needs without this linkage.

when (time):
Only the enterprise owners and planners can think and provide guidance on the business cycles important to the business: the target cycle of microservices and APIs renewal, retirement, or up-gradation is captured in this *viewpoint-concern*.

Capturing these *viewpoint descriptions* from the *microservices perspective* adds to the enterprise's contextual view and enables planning for DevOps, reuse, vision and strategic direction.

B. Business Model (Owner's View)

"Business Model" describes the owner's view of the enterprise. The " business plan " captures what the enterprise does and why it is captured in the "business plan". A business plan describes the value proposition, customer segmentation, and the channels business wants to reach the customers. "For whom are we creating the value" and "which ones of our customer's problems" we are helping to solve are the usual value propositions.

The following stakeholders see the enterprise from this view: shareholders, investors, founders, the board of governors, etc. Below is a detailed description of their *viewpoint-concerns*.

These stakeholders need to look at the conceptual level of the enterprise to see what API and Microservices will be provisioned by the organization, when, and why.

who (people):
This *viewpoint-concern* identifies workflows required to expose microservices to internal or external audiences (organizations) via APIs.

what (data):
This *viewpoint-concern* describes the data elements (at the semantic level) required to design microservices and APIs.

which (selection):

This *viewpoint-concern* deals with prioritization and selecting essential business processes, scenarios, and functions required to provide APIs and microservices.

where (network):
This *viewpoint-concern* captures locations (geographic locations) where specific microservices may reside (for example, the data center location for latency or specific service offering), and APIs may be offered.

how (function):
This *viewpoint-concern* captures and defines which business functions will be automated with what microservice at the business function level. Similarly, this *viewpoint-concern* also defines which business functions will be exposed via APIs. Table I below captures this viewpoint-concern in tabular form.

TABLE I. BUSINESS TAXONOMY OFFERED VIA μ-SERVICES AND APIS

Business Taxonomy	μ-service$_{1..n}$	API$_{1..n}$
Business Function$_1$	x	x
Business Function$_n$		x

why (motivation):
It is imperative to capture and define the motivation and reasons for providing essential business services, which are the target of microservices and APIs. This *viewpoint-concern* captures and documents these.

when (time):
This *viewpoint-concern* captures business cycles for which a specific API or microservice is provided, modified, updated, or expired.

C. System Model (Designer's View)

The systems model is the view of an enterprise from the system designer's perspective (automation perspective). Where the business processes (candidates for automation) are described in microservice terms. At this stage, the designers (system engineers/architects) identify business concepts (entities) on which the microservices will work (use-cases, application components, etc.) or the entities that are used by microservices and APIs. API naming conventions and naming guidelines are documented. The CD and CI pipelines are decided, and tools are selected.

The following stakeholders see the organization from this view: enterprise architects, requirements engineers, project managers, security architects, privacy specialists, internal and external regulators, auditors (internal and external), business continuity & disaster recovery planners, etc. Below is a detailed description of their *viewpoint-concerns*.

who (people):
This viewpoint-concern captures interface requirements for APIs and microservices. A Decision on which microservices will be system-level microservices and which ones will be user-level microservice is made in this viewpoint-concern. This viewpoint-concern may also capture reverse proxy requirements and considerations from the access perspective. This can be captured in a table like a grid, as shown in Table II.

TABLE II. INTERFACE REQUIREMENTS FOR μ-SERVICES AND APIS

Organization →	internal / external	B2B	B2C
μ-service1..n	x	x	
API1..n	x		
reverse proxy		x	x

what (data):
This *viewpoint-concern* captures logical data elements to be used by each microservice and exposed by APIs. This is the point at which the system designers may decide the microservice patterns ("event sourcing," "side-car" pattern, etc. [15]) to be used for persistent data, as shown in Table III.

TABLE III. PERSISTENCE REQUIREMENTS OF μ-SERVICES

Data elements	Pattern	Persistence
μ-service1..n	side-car	x
API1..n	event-sourcing	

which (selection):
Selection and prioritization based on the business value are captured in this viewpoint-concern - Which business processes, scenarios, functions, and logical data elements are required for APIs and microservices. Please note that which quantifies the selection as described in detail by [13].

where (network):
This viewpoint-concern captures decisions for the provision of microservice at different network locations. The decisions to group services in different Kubernetes pods and clusters are captured in this viewpoint-concern. This can be represented in tabular form, as shown in Table IV below. The API Gateway rules are also defined by this viewpoint-concern, as shown in Table V.

TABLE IV. μ-SERVICES DEPLOYMENT SCHEMES AND POLICES

Deployment	POD1..n	Network1..n	Service Mesh
μ-service1..n	.yml files		.yml files
Container1..n	.yml/json	x	.yml files

TABLE V. API DESIGN AND REVERSE PROXY RULES

APIs	Design	API Gateway rules	endpoint to IP
API$_1$	.yml files	rules	mappings
API$_n$	.yml/json	rules	mappings

how (function):
This viewpoint-concern captures business process models for provisioning APIs and microservices. Each microservice can be designed using a different technology stack. This can be captured as a grid, as shown in Table VI below. Similarly, the design parameters - the output, the input and success criteria, error methods, and the tools used to design the APIs can be captured in this viewpoint-concern, as shown in Table VII.

TABLE VI. μ-SERVICES TECHNOLOGY STACK

Tech Stack	Node	Python	GO
μ-service 1	x	x	
μ-service 2	x	x	
μ-service n		x	x

TABLE VII. μ-SERVICES AND API DESIGN DETAILS (CRUD/REST METHODS AND SUCCESS/ERROR CODES

API Messages	Method	Tech	Code
API 1	get, post	OpenAPI [16]	300
API 2	delete	RAML [17]	201
API n	update	Swagger [18]	200

why (motivation):
This *viewpoint-concern* captures business rules specific to microservices and APIs. Moreover, it can be represented in tabular form, or a business rules engine can be used.

when (time):
This view captures business cycles on which a specific API or microservice is provided, modified, updated, or expired and can be captured in tabular form, see Table VIII.

TABLE VIII. μ-SERVICES AND API LIFE CYCLE

Business Cycles	Provide	Delete	Update
μ-service$_{1..n}$	business cycle	x	data
API$_{1..n}$	data	time	x

D. Technology Model (Builder's View)

"Technology Model" describes microservices from a technology, data, and infrastructure perspective. The technology stack and the design decisions are captured in this viewpoint. These decisions play a major role in the CD and CI pipelines. For example, the configuration files (YAML files for containers and Kubernetes), the Software Development Kits (SDKs), and code samples are described in this *viewpoint- concern*. The data transfer mechanisms between microservices (XML, JSON, Protocol Buffers known as gRPC [19], etc.) are also decided and captured in this *viewpoint-concern*.

The following groups of stakeholders see the organization from this view: developers, programmers, designers, DevOps engineers, network engineers, SRE engineers, etc. Below is a detailed description of their *viewpoint-concerns*.

who (people):
This *viewpoint-concern* captures the detailed design decision of the APIs and microservices. The decision on which microservices will be system-level microservices and which ones will be user-level microservice is decided. From the access perspective, reverse proxy rules and considerations are also captured in this *viewpoint-concern*.

what (data):
This *viewpoint-concern* captures details of physical data elements to be used by each microservice and/or exposed by APIs. This is the point the builders provide details on how the data between the microservices will be interchanged (JSON, XML, or gPRC).

which (selection):
This *viewpoint-concern* deals with the selections and prioritization related to physical data elements. Low-level design decisions for the design of microservices and APIs are captured in this *viewpoint-concern*. For example, the choice of containers available for a specific microservice and the priority given to a specific container (for its light-weightiness or availability in the private repositories).

where (network):
This *viewpoint-concern* captures the detailed design of network locations where microservices are deployed. This viewpoint-concern captures system networking requirements, such as Kubernetes clusters and the container networking requirements and needs, as shown in Table IX.

TABLE IX. K8S CLUSTERS NETWORKING

POD$_{1..n}$	POD CNI	IP$_{1..n}$
μ-service$_{1..n}$	IP	IPs
Container$_{1..n}$	IP	IPs

how (function):
This *viewpoint-concern* captures system-level design decisions of the APIs. The technology stack for each microservice is captured, and the design parameters of the APIs - the access, authorization, and authentication mechanisms, are captured in this *viewpoint- concern*.

why (motivation):
This *viewpoint-concern* captures business rules implementations for microservices and APIs.

when (time):
This *viewpoint-concern* captures implementation de- tails for the expiration and deletion of microservices and APIs at specific business cycles and events.

E. Detailed Representations (Consumer's View)

This viewpoint describes how external consumers see the enterprise from the outside. The stakeholders in this group include the business owners of external enterprises, the architects, and the developers' community outside the enterprise. These stakeholders' *viewpoint-concerns* are how to conduct business with the enterprise and how the systems-level integration will work.

The US-based technology giants, also known as FANG (Facebook, Amazon, Netflix, Google) [20], offer SDKs and code examples that directly target this group of stakeholders. The terms like "developers experience" come from providing this *viewpoint-concern*. Most of the industry leaders in technology provide detailed descriptions of their APIs and issue SDKs of their services. Their target audiences are the designers, architects, and developers in consuming organizations (these audiences, in turn, steer the partner organizations' business leaders to buy services from the enterprise). In this *viewpoint-concern*, the needs of external stakeholders are captured and defined. These views and viewpoint-concerns of stakeholders enable other organizations to consume what the enterprise offers.

This view focuses on the detailed representation of the organizations' microservices and APIs - API documentation (Swagger, OpenAPI, GraphQL [21]); and the Microservices SDKs and code samples. These *viewpoint-concerns* can be captured in the form given in Table X.

TABLE X. SDKS AND CODE SAMPLES

Consumers View	SDK	Code Samples	Documents
μ-service$_{1..n}$	details	details	details
API$_{1..n}$	details	details	details

IV. THE PROPOSED FRAMEWORK

Based on the descriptions given in Methods (Section III), we organize viewpoint-concerns in tabular form, as shown in Table XI. The columns in Table XI represent the ordered viewpoints, and the rows are stakeholders' groups drawing from work by [13, 14]. Each cell in Table XI contains viewpoint-concerns of a given stakeholders' group from a

specific viewpoint. Table XI enables capturing stakeholders' viewpoint-concerns for API endpoints and microservices and enables creating an API economy for the Enterprise.

TABLE XI. THE PROPOSED FRAMEWORK: ROWS: IN THE BOLD LETTERS ARE THE VIEW, HOW STAKEHOLDERS VIEW AN ORGANIZATION; AND THE [] LIST KEY STAKEHOLDERS IN THIS GROUP. COLUMNS ARE STAKEHOLDER VIEWPOINTS (IN THE ORDER PROPOSED BY [13, 14]). THE CELLS ARE HIGH-LEVEL STAKEHOLDER CONCERNS FROM A SPECIFIC VIEWPOINT

Microservices (µ-services)	**(1) People (who)**	**(2) Data (what)**	**(3) Selection (which)**	**(4) Network (where)**	**2,3/2,4 $\Rightarrow$ (5) (How)**	**2,5 $\Rightarrow$ (6) (Why)**	**4,5 $\Rightarrow$ (7)**
Scope (Ballpark View): [Business Development Directors, CIOs, CFOs, and CSOs]	list of businesses the enterprise may develop µservices for and expose or consume APIs from.	List of entities required for each µ-service and/or API	Which: prioritized list of organizations, µ-services and APIs candidates for reuse	List of locations for hosting µ-services and related technologies	List of business processes each µ-service performs	List of business goals/strategies for each µ-service and API	List of cycles significant for µ-service and APIs
Business Model (Owners' View): [Shareholders, Investors, Founders, Board of Governors]	e.g., µ-service workflow models	e.g., µ-service semantic model	e.g., Which µservices and APIs are a candidate for reuse	e.g., Datacenter locations where specific µ-service reside	e.g., Business Taxonomy and linkage between Business processes with µservices and APIs, as shown in Table I	e.g., Business goals related to µservices and APIs	e.g., business cycles for when and API or µ-service is provided, modified, updated, or expired.
System Model (Designer's View): [enterprise architects, requirements engineers, security architects, privacy specialists, BCP planners]	e.g., s interface requirements for APIs and µ-services Table II	e.g., logical data elements to be used by each µ-service and exposed by APIs - Table III	e.g., Selection and prioritization of business processes/logical data elements required by API and µ-services	e.g., distributed µ-service and container architecture Table IV, Gateway and reverse proxy rules Table V	e.g., µ-service applications architecture Table VI & Table VII	e.g., µ-service business rule model	e.g., automated deletion/ provisioning of µ-services are pre-specified time
Technology Model (Builder's View): [DevOps engineers, programmers, network engineers, SRE engineers]	e.g., µ-service presentation architecture	e.g., µ-service physical data elements; data interface specifications (JSON, XML or gPRC)	e.g., Selection and prioritization of low-level design components, like containers, and pods etc.	e.g., µ-service and container deployment locations, locations of Kubernetes clusters and configuration files	e.g., µ-service technology stack, API access, authentication and authorization mechanisms	e.g., Business rules implementations for µ-services and APIs	e.g., zero-touch deployment scripts for auto provision/deletions of µ-services
Detailed Representations (Consumer's View):	SDKs, APIs and Code Samples; See Table X e.g., (curl -v -X GET https://api.sandbox.paypal.com/v1/payment-experience/web -profiles/XP-8YTH-NNP3-WSVN-3C76 -H" Content-Type: application/json" -H" Authorization: Bearer Access-Token");						

V. DISCUSSION

Microservices architecture has created an API economy for organizations. Organizations provide services to their internal and external clients via APIs to monetize business services. APIs serve as a contract between the provider and the consumer; therefore, the organizations need to design, provision and manage APIs using some standard methods.

The life cycle of APIs can be divided into the following three broad categories. a) API design, b) API discovery and orchestration, and c) API management. These stages help set the APIs' value and the microservices that enable these APIs.

There are two approaches for API design, Code First and Design First. The Code First approach is a more traditional approach to building APIs. In this approach, the development of the code is followed by the requirements. The documentation and testing of the API endpoints are carried out once the system is already in place. On the other hand, in the Design First approach, the API endpoint design is carried out before any code is written. The latter is a much more structured approach and allows engineering of the stakeholders' needs and requirements. We show that the methods described in Table XI enable capturing requirements and enable engineering of the stakeholder needs for API endpoints.

API discovery and orchestration are made via discovery and enterprise-service-buss (ESB) tools like Kafka [22] and NATS [23]. API management includes consumption analytics, rate limiting, throttling and pricing. Many tools have matured in this domain like google Apigee [24] and Kong [25] etc.

The Design First approach is gaining popularity lately, as the tools are maturing, for example, Swagger [18] and Postman [26]. Another reason for the Design First approach to API design is that the business folks find it quite simple to express their requirements in the form of API access mechanisms, security, data and functionality. The framework developed in Table XI enables organizing it.

VI. EXPERIMENTS: FRAMEWORK APPLICATION

We applied the methods developed in this publication to a cloud-native microservices-based system by an industrial partner organization. This system exposes internal as well as external API endpoints. We show that capturing stakeholder viewpoint-concerns using methods described in this publication enables the better and complete design of APIs and microservices, particularly from the *Design First approach*.

A. *Brief Description of the System*

Our industrial partner designed a system to optimize leaf and spine data centres using dynamic policies by learning traffic loads in a network fabric [27]. These policies are deployed to the network fabric using an SDN controller.

All the system components are designed as microservices, which expose internal and external APIs for communication among the components.

They simulated the leaf and spine data center using Mininet [28] as a microservice and exposed the network traffic via internal endpoints to the SDN controller NOX [29]. The SDN controller is deployed as a separate microservice and monitors the network traffic in the fabric (leaf nodes are used to generate and consume network traffic). This traffic is streamed to the machine learning layer using Kafka [22], with Kafka running as a separate microservice in a docker container. The machine learning subsystem consumes Kafka topics (which are network traffic metrics of a node in the fabric). The knowledge-plane subsystem gathers network forecasts for each node in the leaf and spine data center. It builds dynamic policies which are exposed to the SDN controller via API Endpoints. The SDN controller applies these policies to the data plane via southbound APIs.

B. *Application of the Framework*

We apply the methods described in Section III and Table XI to capture the stakeholder viewpoint- concerns to design microservices and API endpoints for the system described in [27].

This is an interactive process; we elaborate on a viewpoint- concern and develop the microservices and endpoints required to fulfil this viewpoint-concern. Please note that for the space limitations, we have included results for only Row 3, which is the designer's view (system-model).

C. *Designers' View*

1) *[row 3, column 1]:* This *viewpoint-concern* captures the interface requirements for the microservices and the required API endpoints for both internal and external systems. Also, the stakeholders in this group are interested to see which APIs are provided to business-2-business (B2B) and business-2-consumer (B2C) partners. Table XII captures this *viewpoint-concern*.

TABLE XII. INTERFACE REQUIREMENTS FOR µ-SERVICES AND APIS

Organization →	int/ext	B2B	B2C
MinnetService	internal		
/relayNBTraffic/{ip}/metrics	internal		
/consumeDynamicPolicy/{ip}/policy	external	x	
/manageEWNodes/{ips}/policy	external	x	x
/ingestNetworkMetrics/{ips}/metric	internal	x	x
/manageEWNodes/{ips}/policy	internal		
/metricProducer/{ips}/metricProducer	internal		
/eventsBrokerService/{ip}/metric	internal		

2) *[row 3, column 2]:* This *viewpoint-concern* captures the logical data elements required to full fill CURD operations by the microservices and the API endpoints. This *viewpoint-concern* also captures if a specific requirement pattern is required for persistence. For example, the Kafka consumer service is events-based, and the events decide the persistence. The KnowledgePlaneService is where the historical policies are kept and need to be stored in a parallel and promising candidate for the side-car pattern. This is given in Table XIII.

TABLE XIII. PERSISTENCE REQUIREMENTS OF µ-SERVICES

µ-service	Pattern	Persistence
KnowledgePlaneService	side-car	x
ConsumerService	event-sourcing	x

3) *[row 3, column 3]*: This *viewpoint-concern* help prioritize business processes/logical data elements required by API and µ-services. The list of µ-services given in Table XIV results from this prioritization and selection.

TABLE XIV. µ-SERVICES UPDATE AND DELETION CYCLES

µ-service	Provision	Update	Retire
MinnetService	x	mid-year	
SDNService	2022	quarterly	
ProducerService		biannually	x
EventsBrokerService		quarterly	2021/9
ConsumerService		quarterly	x
KnowledgePlaneService	x	quarterly	

4) *[row 3, column 4]*: In this *viewpoint-concern* we capture the network location and the PODs a service is running. The internal ports to POD mappings and the physical disk mount- points are also captured in this viewpoint-concern as shown in Table XV and Table XVI.

TABLE XV. µ-SERVICES DEPLOYMENT SCHEMES AND POLICES

API Endpoint	POD/Servic	Network	Manifest
/metricConsumer/{ip}/metric	mConsumer	vpc1	mconsumer.yaml
/ingestNetworkMetrics/{ips}/metric	netMetrics	vpc1	netMetrics.yaml
/manageEWNodes/{ips}/policy	eastWestNode	vpc2	eastWestNodes.ya
/eventsBrokerService/{ip}/metric	kafkaBroker	vpc2	kafkaBroker.yaml
/metricConsumer/{ip}/metricCons	mConsumer	vpc3	mconsumer.yaml
/zookeeperService{ip}/metric	zookeeper	vpc3	zookeeper.yaml

TABLE XVI. API DESIGN REVERSE PROXY AND PODS

API endpoints	POD	Mount Point	Internal
/metricConsumer/{ip}/metric	mcons-0	/app/consumer	8888
/ingestNetworkMetrics/{ips}/metric	nmets-0	/app/metrics	8703
/manageEWNodes/{ips}/policy	ew-0	/app/opt	8702
/eventsBrokerService/{ip}/metric	brok-0	/app/broker	6666
/zookeeperService{ip}/metric	zook-0	/app/zook	8866

5) *[row 3, column 5]*: This *viewpoint-concern* captures the technology stack for each µ-service and the design mechanisms of the API endpoints as given in Table XVII and Table XVIII.

TABLE XVII. SERVICES TECHNOLOGY STACK

µ-service	Node	Python	GO
MinnetService		x	
SDNService		x	
ProducerService		x	x
EventsBrokerService		x	
ConsumerService		x	
KnowledgePlaneService		x	

TABLE XVIII. μ-SERVICES AND API DESIGN DETAILS (CRUD/REST METHODS, SUCCESS/ERROR CODES AND DESIGN TOOLS/STANDARDS)

API endpoints	Method	Tech	Code
/metricConsumer/{ip}/metric	get, post, update	OpenAPI	200, 201, 202
/ingestNetworkMetrics/{ips}/metric	get, post, patch	OpenAPI	200, 201, 500
/manageEWNodes/{ips}/policy	delete, put	OpenAPI	201, 400, 401, 403
/eventsBrokerService/{ip}/metric	get	OpenAPI	200, 201, 400
/zookeeperService{ip}/metric	get, post	OpenAPI	300, 401

6) *[row 3, column 6]:* In this *viewpoint-concern,* we capture business rules for μ-services. For example, clients with over one year of subscription will get a 10% discount on cloud services.
7) *[row 3, column 7]:* Below is the schedule for provisioning, revision and retirement/deletion of μ-services at pre-specified times. For example, the stakeholders in this group have recommended using a different message broker (NATS) instead of Kafa so that this service will be retired and decommissioned at a predetermined time.

D. Microservice definitions based on the Framework

Following the Design First approach of APIs design, we analyzed and engineered the stakeholders' viewpoint-concerns given in Section VI to design the API endpoints. These API endpoints are then grouped in microservices. We have abstracted the API endpoints to the route level for brevity and privacy of this commercial application. Table XIX provides a list of microservices designed with associated value to the enterprise.

TABLE XIX. μ-SERVICES

	μ-service	Stakeholder	Value
1	MinnetService	CEO, Engineering	$$
2	SDNService	CEO, Marketing, Sales	$$$$
3	ProducerService	Architects, Engineering	$
4	EventsBrokerService	Network, DevOps	$$
5	ConsumerService	Network, Engineering	$$
6	KnowledgePlaneService	AI, CEO, Engineering	$$$

VII. CONCLUSIONS AND FUTURE RESEARCH

In this publication, we used an Enterprise Architecture approach to capture stakeholder viewpoint-concerns for microservices-based cloud-native developments. We described methods to define a new Enterprise Architecture framework for DevOps and agile SDLCs. The order presented in this framework ensures capturing stakeholder views in their entirety [14]. We applied this framework to an industrial system to design APIs and microservices. We showed that the methods described in this publication are an effective tool for monetizing the APIs and microservices by associating business value to these. This framework enables the creation of taxonomies of microservices and APIs, which can help configure API management tools like [24, 30, 25] to prescribe rate limits, set value and monetize.

With the advent of microservices-based cloud-native developments, organizations face another challenge, known as BiModel IT[1]. Most legacy systems may fulfil their functional requirements; these are not scalable or interoperable with the modern microservices-based architectures.

The framework enables dealing with BiModal IT challenges by providing tools to bridge the gap between these two disjoint worlds. The framework opens more doors for research and offers practitioners new insights. We intend to use graph theory to cluster stakeholders' *viewpoint-concerns* and microservices.

REFERENCES

[1] C. M. Pereira and P. Sousa, “A method to define an enterprise architecture using the Zachman framework,” in Proceedings of the 2004 ACM symposium on Applied computing. ACM, 2004, pp. 1366–1371.

[2] H. Shah and M. El Kourdi, “Frameworks for enterprise architecture,” It Professional, vol. 9, no. 5, pp. 36–41, 2007.

[3] R. Winter and R. Fischer, “Essential layers, artifacts, and dependencies of enterprise architecture,” in Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW'06. 10th IEEE International. IEEE, 2006, pp. 30–30.

[4] S. A. Bernard, An introduction to enterprise architecture. Author House, 2012.

[5] M. Lankhorst, Enterprise architecture at work. Springer, 2009, vol. 352.

[6] J. Schekkerman, How to survive in the jungle of enterprise architecture frameworks: Creating or choosing an enterprise architecture framework. Trafford Publishing, 2004.

[7] S. Leist and G. Zellner, “Evaluation of current architecture frameworks,” in Proceedings of the 2006 ACM symposium on Applied computing. ACM, 2006, pp. 1546–1553.

[8] A. Tang, J. Han, and P. Chen, “A comparative analysis of architecture frameworks,” in Software Engineering Conference, 2004. 11th Asia-Pacific. IEEE, 2004, pp. 640–647.

[9] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, “Viewpoints: A framework for integrating multiple perspectives in system development,” International Journal of Software Engineering and Knowledge Engineering, vol. 2, no. 01, pp. 31–57, 1992.

[10] N. Rozanski and E. Woods, Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley, 2012.

[11] J. A. Zachman, “A framework for information systems architecture,” IBM systems journal, vol. 26, no. 3, pp. 276–292, 1987.

[12] L. N. Flint, Newspaper writing in high schools: Containing an outline for the use of teachers. Pub. from the Department of journalism Press in the University of Kansas, 1917.

[13] M. Sultan and A. Miranskyy, “Ordering interrogative questions for effective requirements engineering: The w6h pattern,” in 2015 IEEE Fifth International Workshop on Requirements Patterns (RePa). IEEE, 2015, pp. 1–8.

[14] Sultan, Mujahid, and Andriy Miranskyy. "Ordering stakeholder viewpoint concerns for holistic enterprise architecture: the W6H framework." In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, pp. 78-85. 2018.

[1] Gartner defines BiModal IT as “the practice of managing two separate, coherent modes of IT delivery, one is focused on stability and the other on agility”

[15] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for mi- microservices: a systematic mapping study." SCITEPRESS, 2018.
[16] (2020, Sep) Openapi. [Online]. Available: https://www.openapis.org/
[17] (2020, Sep) Raml. [Online]. Available: https://raml.org/
[18] (2020, Sep) Swagger. [Online]. Available: https://swagger.io
[19] (2020, Sep) grpc. [Online]. Available: https://grpc.io/
[20] D. Winseck, "The geopolitical economy of the global internet infrastructure," Journal of Information Policy, vol. 7, pp. 228–267, 2017.
[21] (2020, Sep) Graphql. [Online]. Available: https://graphql.org/
[22] Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." In Proceedings of the NetDB, vol. 11, pp. 1-7. 2011.
[23] (2021, Oct) nats. [Online]. Available: https://nats.io/ [24] (2021, Aug) Apigee. Https://docs.apigee.com/.
[25] (2021, Aug) kong. Https://konghq.com/products/kong-enterprise/. [26] (2021, Oct) Postman. [Online]. Available: https://www.postman.com/
[27] M. Sultan, D. Imbuido, K. Patel, J. MacDonald, and K. Ratnam, "Designing knowledge plane to optimize leaf and spine data center," in 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), 2020, pp. 13–15.
[28] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in Proceedings of the 8th international conference on Emerging networking experiments and technologies, 2012, pp. 253–264.
[29] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 3, pp. 105– 110, 2008.
[30] (2021, Aug) Ibm api connect. Https://www.ibm.com/cloud/api-connect.